

Abschlussprüfung Sommer 2022

Ausbildungsberuf

Fachinformatiker Anwendungsentwicklung

Projektdokumentation

Implementierung eines neuen Systems zur Erfassung, Pflege, Verwaltung von
Versandverpackungen und dessen Zuordnung zu hergestellten Produkten



Herr Sandro Schmitt

Identnummer: 815430

E-Mail: schmittsandro40@gmail.com

Telefon: +49 9391 913858

Ausbildungsbetrieb: FLYERALARM Industrial Print GmbH

Projektbetreuer: Herr Marco Thiergart

E-Mail: marco.thiergart@flyeralarm-industrialprint.com

Telefon: +49 9391 50314902



Inhaltsverzeichnis

1	Einleitung	5
1.1	Projektumfeld	5
1.2	Projektbeschreibung	5
1.3	Projektbegründung	6
1.4	Projektschnittstellen	6
1.5	Projektabgrenzung	6
2	Projektplanung	6
2.1	Projektphasen	6
2.2	Ressourcenplanung	7
2.3	Entwicklungsprozess	7
3	Analyse	7
3.1	Ist-Analyse	7
3.2	Wirtschaftlichkeitsanalyse	8
3.2.1	Projektkosten	8
3.2.2	Gründe für das Projekt	8
3.3	Soll-Konzept	8
4	Entwurf	9
4.1	Zielplattform	9
4.2	Datenbankstruktur	9
4.3	Mockup Benutzeroberfläche	9
5	Implementierung	10
5.1	Erstellen der Datenbank	10
5.2	Erstellen der REST API	10
5.2.1	Verwendete Pakete	10
5.2.2	Datenbankanbindung	10
5.2.3	Routenimplementierung	11
5.2.4	Authentifizierung	11
5.2.5	Schnittstellendokumentation mit Swagger	12
5.3	Erstellen des Dashboards	13
5.3.1	Verwendete Pakete	13
5.3.2	Anbindung an Rest API	13
5.3.3	Erstellen der Benutzeroberfläche	13
5.3.4	Erstellen der Authentifizierung	14
5.4	Erstellen des NuGet Pakets	14
5.4.1	Verwendete Pakete	14
5.4.2	Implementierung der Logik	14
5.4.3	Erstellen der Benutzeroberfläche	15

6	Testen und Abnahme	16
6.1	Test durch den Entwickler	16
6.1.1	Code-Review	16
6.1.2	Schreibtischtests	16
6.1.3	Debugging	16
6.2	Abnahme durch Teamleiter	17
7	Dokumentation	17
7.1	REST-Schnittstelle	17
7.2	Dashboard	17
7.3	NuGet-Paket	17
8	Fazit	17
 Glossar		I
 Abkürzungsverzeichnis		II
 Anhangsverzeichnis		
Anhang 1: Projektphasen		IV
Anhang 2: Datenbankmodell		V
Anhang 3: Mockup Dashboard		VI
Anhang 4: Ressourcenplanung		VI
Anhang 5: Ausschnitt Swagger-Dokumentation		VII
Anhang 6: Finales Design des Dashboards im Darkmode		VII
Anhang 7: Anwenderdokumentation Dashboard		VIII
Anhang 8: Anwenderdokumentation NuGet Umfrage		VIII
 Tabellenverzeichnis		
Tabelle 1: Projektphasen		7
Tabelle 2: Kostenverteilung		8
 Abbildungsverzeichnis		
Abbildung 1: Befehl zum Erstellen einer Migration		10
Abbildung 2: FluentAPI eines Unique Constraints		11
Abbildung 3: Route zum Ändern der Verpackungen eines Produkts		11
Abbildung 4: Verwendung des Authorize Tag		12

Abbildung 5: Request, um alle Packages zu erhalten	13
Abbildung 6: ViewModel	16

1 Einleitung

Zur besseren Übersichtlichkeit werden unterstrichene Wörter im Glossar genauer erläutert. Die Langformen der Abkürzungen sind im Abkürzungsverzeichnis zu finden.

1.1 Projektumfeld

Die Flyeralarm GmbH (im Folgenden FA genannt) ist eine der größten deutschen E-Commerce-Unternehmen und eine der führenden Online-Druckereien Europas im B2B Bereich. 2002 wurde diese als Ein-Mann Unternehmen gegründet und beschäftigt mittlerweile mehr als 2000 Mitarbeiter. Das Produktsortiment umfasst drei Millionen Produkte und wird stetig erweitert. Dazu gehören zum Beispiel Flyer, Visitenkarten, Kalender, Magazine und viele weitere Artikel. FA betreibt den Onlineshop und ist für die Datenannahme, Bearbeitung der Kundenaufträge und das Erstellen von Sammelbögen zuständig. Ihre Tochtergesellschaft, FLYERALARM Industrial Print GmbH (im Folgenden FAIP genannt), übernimmt neben dem Drucken im Sammeldruckverfahren auch die Weiterverarbeitung, Montage, Konfektionierung und die Vorbereitung für den Versand. Sie ist mit acht Standorten in Deutschland vertreten und beschäftigt ca. 1200 Mitarbeiter. Das Kerngeschäft der IT-Abteilung ist hierbei das Aufarbeiten der Druckdaten von FA, damit diese den Mitarbeitern in der Produktion und den Druckmaschinen zur Verfügung gestellt werden können. Hier absolviert der Autor seine Ausbildung.

1.2 Projektbeschreibung

Wie bereits in 1.1 erwähnt, werden die Druckdaten von FA auf Sammelbögen platziert und an FAIP übergeben. Die Druckdaten werden zusammen mit Metadaten in einem ZIP-Archiv auf einem FTP Server zur Verfügung gestellt. Daten werden mithilfe eines Windows-Services heruntergeladen und von verschiedenen Programmen für die Produktion aufbereitet. Unter anderem werden hier auch die benötigten Verpackungen auf Auftragsebene berechnet. Die Berechnung erfolgt unter Berücksichtigung vieler Parameter, welche sowohl aus einer relationalen Datenbank als auch aus anderen Quellen bezogen werden müssen. Bei einem Auftrag werden so zum Beispiel die Auflage, Stärke, Größe und Gewicht des verwendeten Papiers und des Kartons, das Maximalgewicht des Paketdienstleisters und ob ein Rollenkern verwendet werden kann, berücksichtigt. Die Berechnung erfolgt allerdings nur, wenn keine Ausnahme in der Datenbank hinterlegt ist. Die Anzahl und Art der Verpackungen werden hierbei anhand der Grammatik und Auflage festgehalten. Ist eine solche Ausnahme vorhanden, wird direkt der hinterlegte Wert verwendet, ohne die Berechnung durchzuführen. Dadurch ist es nicht möglich, den neuen Anforderungen des Verpackungsschutzgesetzes gerecht zu werden. Das Ziel dieses Projektes ist es, jeder Produktvariante genau eine passende Verpackungsmöglichkeit zuzuordnen. Das Projekt soll dabei aus 4 Teilen bestehen. Ein NuGet Paket, wodurch die Daten per Crowdsourcing erfasst werden können, eine relationale

Datenbank zum Speichern der erfassten Daten, eine REST Schnittstelle und ein Dashboard zum Auslesen, Bearbeiten und Löschen der Daten.

1.3 Projektbegründung

Durch die Umsetzung des Projektes können die Daten in Zukunft zuverlässiger erfasst und ausgewertet werden. Durch das Crowdsourcing wird ein großer Teil des Wartungsaufwandes in die Verantwortung der Mitarbeiter übertragen und es wird sichergestellt, dass die erfassten Verpackungen aktuell und produktionsnah sind. Dadurch ist es möglich, dem neuen Verpackungsschutzgesetz im vollen Maße gerecht zu werden und die genaue Menge der verwendeten Verpackungen zu bestimmen.

1.4 Projektschnittstellen

Die Anwendung kommuniziert mit einer externen Schnittstelle, um zu einer Auftragsnummer die dazugehörige Produktvarianten-ID und die Auflage zu bestimmen.

1.5 Projektabgrenzung

Das Projekt beinhaltet nicht die Beschaffung der Produktdaten. Diese werden von einem Importer bei Dateneingang in die Datenbank geschrieben. Auch das Aufsetzen und Anlegen der Datenbank und des Datenbankbenutzers sind nicht Teil des Projektes. Diese waren schon vorhanden.

2 Projektplanung

2.1 Projektphasen

Das Projekt wurde innerhalb von 70 Stunden umgesetzt. Die Gesamtzeit wurde auf sechs Phasen aufgeteilt. In der Tabelle 1 sieht man einen Vergleich der geplanten Stunden des Projektantrags gegenüber den wirklich verwendeten Stunden. Im Gegensatz zur geplanten Zeit wurde weniger Zeit für die Analyse und Implementierung benötigt, allerdings wurden diese Stunden dann in der Einführungs- und Dokumentationsphase beansprucht. Dadurch wurde lediglich die Aufteilung untereinander verändert, während die Gesamtzeit des Projektes unverändert bleibt. Eine detaillierte Aufteilung der Phasen ist im Anhang 1 zu finden.



Projektphase	Stunden geplant	Stunden verwendet	Differenz
1. Analyse	5	3	-2
2. Entwurf	4	4	
3. Implementierung	41	38	-3
4. Test	5	5	
5. Einführung	3	4	+1
6. Dokumentation	12	16	+4
Summe:	70	70	

Tabelle 1: Projektphasen

2.2 Ressourcenplanung

Die verwendete Hard- und Software war bereits vorhanden oder wurde kostenfrei zur Verfügung gestellt. Weitere verwendete Ressourcen können dem Anhang 4 entnommen werden.


2.3 Entwicklungsprozess

Bei der Entwicklung wurde sich am Wasserfallmodell orientiert. Das sieht man auch an den einzelnen Phasen des Modells. Lediglich die Wartung wurde in diesem Projekt nicht umgesetzt, da diese außerhalb des Projektrahmes liegt. Zuerst wurden die Anforderungen zusammengetragen und besprochen. Anschließend wurde das Datenbankmodell und das Mockup der Seite entworfen, mit dessen Hilfe in der Implementationsphase die drei Teile des Projektes realisiert wurden. Sowohl während als auch nach der Implementierung wurden Tests wie z.B. die Schreibtischtests durchgeführt.

3 Analyse

3.1 Ist-Analyse



Vor der Umsetzung des Projektes wurde die Paketgröße und Anzahl für jeden Auftrag individuell bei Eingang der Daten durch eine alte Konsolenanwendung berechnet. Dabei wurden unzählige Faktoren berücksichtigt, wie z.B.: Auflage, Stärke, Größe und Gewicht des Papiers und des Kartons, das Maximalgewicht des Paketdienstleisters und ob ein Rollenkern verwendet werden kann. Außerdem unterschied sich die Berechnung für jeden Standort, da nicht alle Kartonagen an jedem Werk verwendet wurden. All diese Informationen wurden in MSSQL-Datenbanken gespeichert und mussten regelmäßig manuell gepflegt werden. Existierten Sonderfälle zu einem Produkt, wurden die Kartonagen nicht berechnet, sondern der hinterlegte Karton verwendet. Da mittlerweile schon ein großer Teil der Kartonagen über den „Workaround“ abgebildet wurde, war die Berechnung nicht mehr praxistauglich. Außerdem gab es durch die Änderung des Verpackungsgesetzes nun zusätzliche gesetzliche

Anforderungen an die Erfassung verwendeter Verpackungen, was durch die unterschiedlichen Berechnungen und Ausnahmen an den Standorten nicht abbildbar war. Hierfür war es nötig, dass eine einheitliche Verpackungsrichtlinie geschaffen wurde. 

3.2 Wirtschaftlichkeitsanalyse

Da es sich hierbei nicht um ein Projekt mit Gewinnerzielungsabsicht handelt, wird auf eine Amortisationsrechnung verzichtet. Bei den angegebenen Personalkosten dürfen keine echten Werte verwendet werden, deshalb werden diese vom Autor angenommen. Sie sollen lediglich das Verhältnis der verschiedenen Positionen und eine grobe Schätzung der Kosten aufzeigen.

3.2.1 Projektkosten

Wie oben erwähnt, handelt es sich hierbei nicht um die realen Personalkosten. Die Gesamtkosten setzten sich aus den Kosten des Personals und der Ressourcen zusammen. Zu den Ressourcen gehört zum Beispiel, der Strom und die Miete der Büros. Dafür wird ein Pauschalbetrag pro Stunde von 15 € verwendet. Für den Stundenlohn des Personals wurde sich online an dem durchschnittlichen Gehalt des jeweiligen Berufs orientiert. So wird der Auszubildenden im 3. Lehrjahr mit 7 €, ein festangestellter Mitarbeiter des Teams mit 15 €, der Product Owner mit 25 € und der Teamleiter mit 35 € pauschalen Stundenlohn angenommen. Die Gesamtkosten sind der Tabelle 2 zu entnehmen. Hieraus wird gut ersichtlich, dass die höchsten Kosten des Projektes durch den Autor selbst entstanden sind.  

Personal	Zeit	Personalkosten pro h	Personalkosten gesamt
Auszubildender	70 h	7 € + 15 € = 22 €	1.540 €
Teammitglied	2 h	15 € + 15 € = 30 €	60 €
Product Owner	2 h	25 € + 15 € = 40 €	80 €
Teamleiter/ Projektbetreuer	2 h	35 € + 15 € = 50 €	100 €
Gesamtkosten			1.780 €

Tabelle 2: Kostenverteilung

3.2.2 Gründe für das Projekt

Wie schon in 3.2 erwähnt, wird das Projekt aus nicht monetären Gründen umgesetzt. Ein Punkt für die Umsetzung sind die zusätzlichen gesetzlichen Anforderungen, welche im Zuge der Änderungen des Verpackungsschutzgesetzes erfüllt werden müssen. Außerdem verringert sich durch das Projekt der aufwändige Wartungsaufwand in der Datenbank und der komplexe, unverständliche Code zur Berechnung der passenden Verpackung entfällt.

3.3 Soll-Konzept

Die Berechnung soll durch eine direkte Referenzierung einer Kartonage zu einer Produktvariante in einer Datenbank ersetzt werden. Diese sollen mithilfe einer neuen Webseite

für ausgewählte Nutzer einsehbar und anpassbar sein, damit kein Datenbankclient verwendet werden muss. Zur einfacheren Datenpflege, soll außerdem eine Erweiterung entwickelt werden, welche anschließend in die bestehenden Softwarelösungen der Produktion eingebunden werden soll. Mithilfe dieser soll ein Teil der Pflegearbeit per Crowdsourcing zu den Produktionsmitarbeitern ausgelagert werden, um so per Zufallsprinzip fehlende Verpackungsinformationen, mit realitätsnahen Infos aus der Produktion, zu ergänzen.



4 Entwurf

4.1 Zielplattform

Wie in 1.2 erwähnt, besteht das Projekt aus 4 Teilen. Für das Projekt wurden die Sprachen C# und JavaScript verwendet. Damit folgt der Autor dem Standard der FLYERALARM Industrial Print GmbH. Bei der Datenspeicherung wurde auf eine MSSQL-Datenbank gesetzt. Der Autor wählte eine relationale Datenbank, da es sich hierbei um gut strukturierte und gleichbleibende Daten handelt. Das wird deutlich, wenn man das Datenbankmodell (Anhang 2) betrachtet. Daten wie die Verpackung wiederholen sich häufig und können in eine extra Tabelle ausgelagert werden. Durch die feste Struktur ist eine relationale Datenbank im Gegensatz zu einer NoSQL Datenbank besser geeignet, da eindeutige Strukturen zu ihren Stärken zählen. Bereitgestellt wird die Datenbank über einen Windowsserver. Das Erweiterungspaket, oder auch NuGet Paket, wurde als Windows Presentation Foundation (WPF) Klassenbibliothek unter Verwendung von .NET 6.0 entwickelt. Bei der REST-Schnittstelle handelt es sich um eine ASP .NET Core 6.0 Web API. Für das Dashboard wurde die populäre JavaScript Library React verwendet, welche das schnelle und einfache Erstellen von Komponentenbasierten Benutzeroberflächen ermöglicht. Die Webseite und die API werden auf einem Docker Swarm zur Verfügung gestellt. Das NuGet Paket wird über einen eigenen NuGet Store bereitgestellt.



4.2 Datenbankstruktur

Vor dem Erstellen der eigentlichen Datenbank wurde zuerst ein Datenbankmodell entworfen. Dazu wurde die kostenfreie Websteite DbDesigner verwendet. Hierbei wurde darauf geachtet, dass die 3. Normalform eingehalten wird und dass die Beziehungen aller Tabellen untereinander richtig abgebildet werden. Um mit dem Design zukunftsicher zu sein, wurde das Modell mit dem Teamleiter besprochen und abgenommen.



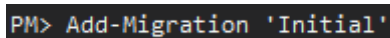
4.3 Mockup Benutzeroberfläche

Nachdem sich der Autor erste Gedanken zum Aussehen der React-Seite gemacht hatte, wurde vor der eigentlichen Erstellung der Webseite ein Mockup der Benutzeroberfläche ausarbeitet. Dabei wurde die kostenfreie Progressive Web App (PWA) Excalidraw verwendet. Nach Absprache mit dem Product Owner wurde das Design finalisiert.

5 Implementierung

5.1 Erstellen der Datenbank

Wie in 1.5 schon erwähnt, wurden die Datenbank und ein passender Benutzer bereits vor dem Projekt angelegt, wodurch der Autor lediglich die Tabellen anlegen musste. Diese wurden mithilfe des Pakets ‚Entity Framework Core‘ (EF Core) von Microsoft unter Verwendung des „Code First“ Ansatzes erstellt. Dazu wird im Gegensatz zum „Database First“ Ansatz zuerst die komplette Datenbankstruktur im Programm mit Klassen modelliert und dann daraus die entsprechende Struktur erstellt. Durch die Verwendung des ORM müssen keine SQL-Befehle geschrieben werden, da diese automatisch durch das Framework generiert werden. EF Core verwendet hierbei den Ansatz von Migrationen, um das Datenbankschema inkrementell an das Datenmodell der Anwendung anzupassen. Um die Änderungen auf die Datenbank anwenden zu können, stellt Microsoft das Paket ‚Entity Framework Core Tools‘ zur Verfügung. Damit können unter anderem neue Migrationen hinzugefügt, entfernt und die Änderungen auf die Datenbank angewandt werden. Insgesamt besteht die Datenbank aus 6 Tabellen. Siehe Anhang 2.



```
PM> Add-Migration 'Initial'
```

Abbildung 1: Befehl zum Erstellen einer Migration

5.2 Erstellen der REST API

5.2.1 Verwendete Pakete

Bei der Entwicklung der API wurden vom Autor weitere Pakete über den NuGet-Manager installiert. Es wurden Pakete zur Authentifizierung des Benutzers gegen den Active Directory Server (Novell.Directory.Ldap.NETStandard) und zum Generieren und Authentifizieren des JWT Tokens (Microsoft.AspNetCore.Authentication.JwtBearer) verwendet. Zur Anbindung an die SQL Datenbank (Microsoft.EntityFrameworkCore.SqlServer) und das Erstellen und Verwalten von Migrationen (Microsoft.EntityFrameworkCore.Tools) wurde das Entity Framework Core verwendet. Für die Dokumentation der Schnittstellen wird Swagger eingesetzt (Swashbuckle.AspNetCore und Swashbuckle.AspNetCore.Newtonsoft). Diese Pakete waren dem Autor bereits von vorherigen Projekten bekannt und der Umgang mit ihnen musste nicht erlernt werden.

5.2.2 Datenbankanbindung

Auf die Anbindung an die Datenbank wurde bereits teilweise in 5.1 eingegangen. Zuerst wurden hierbei die Models der Tabellen als Klassen angelegt. Diese Klassen werden dann im ‚DbContext‘ als DbSet verwendet, wobei ein DbSet eine Tabelle in der Datenbank widerspiegelt. Die Klasse implementiert das Interface DbContext von EF Core und überschreibt dabei drei Methoden. ‚OnModelCreating‘ wird beim Erstellen des

Datenbankmodels aufgerufen. Hier werden weitere Eigenschaften mithilfe der Fluent API, wie Many-to-Many Beziehungen und Unique Constraints, abgebildet.

```
modelBuilder.Entity<Product>()
    .HasIndex(x => new { x.Quantity, x.ProductVariantId })
    .IsUnique();
```

Abbildung 2: FluentAPI eines Unique Constraints

Die zwei Methoden zum Speichern wurden erweitert, damit automatisch der Zeitpunkt des Erstellens und des Bearbeitens eines Datensatzes in die Tabelle eingefügt werden. Der ganze ‚DbContext‘ wird dann als Service in der Startup Datei eingebunden, damit dieser in jedem Controller, per Dependency Injection verwendet werden kann.

5.2.3 Routenimplementierung

Als nächstes wurden die einzelnen Controller und Routen entworfen. Dabei sind sieben Controller entstanden, von denen sechs die einzelnen Tabellen in der Datenbank repräsentieren und einer für die Authentifizierung des Nutzers zuständig ist. Hierbei wurden für alle Datencontroller immer zuerst CRUD-Routen angelegt und diese dann nachträglich an die genauen Bedürfnisse angepasst. Erweitert werden musste hierbei der ‚ProductController‘. Dieser besitzt neben den normalen Routen noch zwei weitere Routen. ‚survey‘ gleicht anhand von ProductVariantId, Quantity und UserId in der Datenbank ab, ob zu diesem Product noch Umfragen durchgeführt werden müssen. Dem zweiten Endpunkt kann in der URL eine Id eines Products und im Body ein Package Array übergeben werden. Damit werden dann die hinterlegten Daten zu diesem Product überschrieben (siehe Abbildung 3).

```
[HttpPatch("{id}/packages")]
0 Verweise
public async Task<IActionResult> PatchPackage([FromRoute] int id, [FromBody] ProductPackage[] packages)
```

Abbildung 3: Route zum Ändern der Verpackungen eines Produkts

Eine andere Erweiterung wurde im ‚ProductPackageCrowdController‘ durchgeführt. Hier wurde die Standard POST Prozedur angepasst. Sobald zu einem Product die zehnte Umfrage durchgeführt wurde, wird automatisch die am häufigsten verwendete Antwort ermittelt und als ProductPackage zum Product gespeichert.

5.2.4 Authentifizierung

Als letztes wurde die Authentifizierung implementiert. Dafür wurde ein ‚AuthService‘ entwickelt, welcher den Benutzernamen und das Passwort gegen den Active Directory Server authentifiziert und danach einen JWT Token generiert. Dieser wird dann zusammen mit den Benutzerdaten zurückgegeben. Außerdem kann der Service einen bereits erstellten Token validieren und falls dieser noch nicht abgelaufen ist, einen neuen mit samt der Benutzerdaten zurückgeben. Diese Methoden werden nur an einer Stelle im Code aufgerufen, dem

„AuthController“. Dieser besitzt zwei Endpunkte. Die „login“ Route bekommt den Benutzernamen und das Passwort in einem Model übergeben und ruft damit die entsprechende Methode des Service auf. Dieser gibt dann einen User zurück. Falls dieses Objekt „null“ ist, wird der Statuscode 401 (Unauthorized) mit der Fehlermeldung „Benutzername oder Passwort ist falsch“ zurückgegeben. Ansonsten wird der User ohne Passwort als Response übermittelt. Die zweite Route „renew“ nimmt den alten Token im Body, prüft und generiert diesen über die Methode des Service neu und gibt dann auch den Benutzer zurück. Neben der Authentifizierung mit Benutzername und Passwort kann man sich an der API aber auch mithilfe eines API Keys authentifizieren, damit auch andere Programme, wie das NuGet-Paket, ohne Benutzerlogin auf die REST API zugreifen können. Dafür wurde die „AuthMiddleware“ Klasse erstellt, welche das Interface „IAuthorizationHandler“ implementiert. Hier wird versucht, den Header „X-API-KEY“ des Requests auszulesen und diesen mit dem hinterlegten Original zu vergleichen. Sind beide gleich, wird die Anfrage ausgeführt, ansonsten erfolgt eine Response mit dem Statuscode 401 (Unauthorized). Diese Abfrage erfolgt allerdings nur, falls kein „Authorization“ Header, also kein Bearer Token, gesetzt ist. Ansonsten wird dieser verwendet. Sobald ein Controller jetzt mit dem Attribut „Authorize“ markiert wird, wird vor der Abfrage automatisch geprüft, ob eine der beiden oben erwähnten Anforderungen zutrifft.

```
[Authorize]
[Route("[controller]")]
[ApiController]
1 Verweis
public class ProductController : ControllerBase
```

Abbildung 4: Verwendung des Authorize Tag

5.2.5 Schnittstellendokumentation mit Swagger

Swagger verwendet Kommentaren aus dem Code, um damit automatisch eine Weboberfläche zu erstellen, über welche grafisch alle Routen und deren Parameter angezeigt und getestet werden können. Damit Swagger die Schnittstellen richtig dokumentiert und auch die Authentifizierungsarten in der Weboberfläche angezeigt werden, mussten noch einige Dinge in der Startup Datei angepasst werden. Dazu wurden die zwei Arten als SecurityDefinition angelegt und die SecurityRequirements festgelegt. Anschließend wurde die API-Dokumentationsdatei mit den Kommentaren aus den Controllern eingebunden, damit diese von Swagger ausgelesen werden kann. Danach wurden die Kommentare zu den einzelnen Routen angelegt. Ein Ausschnitt aus der Swagger-Dokumentation ist in Anhang 5 zu finden.

5.3 Erstellen des Dashboards

5.3.1 Verwendete Pakete

Wie auch schon bei der REST API, wurden auch bei der Webseite noch zusätzliche Pakete nachinstalliert. Hierbei wurde der Paketmanager npm verwendet. So wurde für das Design der Webseite eine Komponentenbibliothek (mantine) und eine Iconsammlung (tabler-icons-react) verwendet. Zum Anzeigen der Daten in einer Tabelle wurde außerdem ein Paket installiert, welches die Erstellung vereinfacht (react-table). Auch das Routing über mehrere Seiten wurde mithilfe eines Paketes umgesetzt (react-router-dom). Auch eine Sammlung nützlicher Funktionen (lodash) und ein Paket zum Absetzen von asynchronen REST Request (axios) wurde verwendet. Alle Pakete waren dem Autor durch vorherige Web-Projekte bekannt.

5.3.2 Anbindung an Rest API

Als erstes wurde die Anbindung an die API erstellt. Hierbei wurden fünf Funktionen angelegt und in eine eigene Datei ausgelagert, welche jeweils einen Request an einen Endpunkt der API tätigen. Ein Beispiel ist hierfür in Abbildung 5 zu sehen.

```
const getPackages = (jwt) => axios.get(`${process.env.API_URL}/package`, { headers: { Authorization: `Bearer ${jwt}` } })
```

Abbildung 5: Request, um alle Packages zu erhalten

So gibt es für die Authentifizierung eine ‚login‘ und ‚renew‘ Funktion. Die anderen drei sind für das Abrufen der Verpackungen und Produkte und das Überschreiben der verwendeten Verpackungen zu einem Produkt. Dabei setzen alle Requests, bis auf die ‚login‘ Route, einen ‚Authorization‘ Header mit dem Bearer Token, um sich an den Endpunkten zu authentifizieren.

5.3.3 Erstellen der Benutzeroberfläche

Danach wurde mit der Erstellung der Benutzeroberfläche begonnen. Wie schon in 5.3.1 erwähnt, wurde dafür eine Bibliothek mit Komponenten verwendet. Außerdem wurde als Orientierung das zuvor erstellte Mockup benutzt, um die Seite nachzubauen. Um hierbei Datenredundanz zu vermeiden, wurden einzelne Komponenten wie z.B. der Header jeweils in eine eigene Datei ausgelagert. Betritt ein Benutzer die Webseite, wird er von der Login Seite begrüßt. Hier kann er sich mithilfe seiner Windows-Anmeldedaten einloggen. Bei korrekter Eingabe der Daten, wird er an das Dashboard weitergeleitet. Hier werden nun in einer Tabelle, alle Produkte aus der Datenbank übersichtlich dargestellt. Die Ansicht kann mithilfe von Wortfiltern für die einzelnen Werte gefiltert und sortiert werden. Außerdem ist eine Zusammenfassung der Daten auf Seiten vorhanden, um sich bequem immer nur einen Teil der Daten anschauen zu können. In dieser Tabelle stehen die ProductVariantId, die Quantity, die QuantityId, die ProductGroupId und der Status der Umfragen eines Produktes. Über einen Knopf können die Hinterlegten Verpackungen angesehen und bearbeitet werden. Hierfür öffnet sich ein Modal. In diesem Modal werden alle ausgewählten Verpackungen



untereinander angezeigt. Über einen Klick auf den Knopf ‚Neue Verpackung‘, kann der Liste eine weitere Verpackung hinzugefügt werden. Hierbei werden jeweils die Anzahl und Art der Verpackung angegeben. Über einen Klick auf den Mülleimer der jeweiligen Zeile, wird diese gelöscht. Sobald die Bearbeitung fertig ist, können mit dem Knopf ‚Absenden‘ die Daten an die Datenbank übermittelt werden. Das Design ist hierbei durchweg schlicht und übersichtlich gehalten, um es dem Benutzer möglichst leicht zu machen. Um die Seite an die jeweilige Vorliebe des Users anzupassen, kann im Benutzermenü, in der oberen rechten Ecke, zwischen dem Dark und Light Mode umgeschaltet werden. Außerdem kann man sich hier auch von der Seite abmelden. Ein Screenshot des fertigen Dashboards ist in Anhang 6 zu finden.

5.3.4 Erstellen der Authentifizierung

Bestimmte Seiten wie das Dashboard, sind nur nach einer validen Authentifizierung des Benutzers abrufbar. Hierfür wurde ein Hook erstellt, mit dem es möglich ist, alle relevanten Funktionen für die Authentifizierung überall in der App aufzurufen. Nach Eingabe der Benutzerdaten und dem Versuch der Anmeldung, werden die Daten an die API übermittelt. Sind diese korrekt, wird das zurückgegebene Objekt im SessionStorage des Browsers gespeichert. Dieses Objekt enthält Informationen über den Benutzer und den JWT. Dieser wird bei jeder Anfrage an die API mitgesendet. Wird auf eine andere Seite navigiert, wird die ‚renew‘ Route mit dem Token aufgerufen. Ist der Token abgelaufen, wird der Benutzer automatisch ausgeloggt und auf die Login Seite navigiert, indem die Daten aus dem SessionStorage entfernt werden. Ist der Token noch valide, wird ein neues Objekt in der Response zurückgegeben, welches dann wiederum die alten Daten überschreibt. Damit ist sichergestellt, dass der Benutzer authentifiziert ist.

5.4 Erstellen des NuGet Pakets

5.4.1 Verwendete Pakete

Auch beim NuGet Paket selbst wurden weitere Pakete installiert. Diese Sorgen für das einheitliche Design der WPF Anwendung (MahApps.Metro), eine zuverlässigere Handhabung mit Json (Newtonsoft.Json) und einfachere und übersichtlichere REST Requests (RestSharp). Neben den externen wurde aber auch ein Inhaus entwickeltes Paket verwendet. Dieses enthält einige wichtige Klasse für die Abfrage der Daten (FAIP.LIB.PrintTalk).

5.4.2 Implementierung der Logik

Als erstes wurde die Klasse PackageService angelegt. Diese enthält eine einzige öffentliche asynchrone Methode ‚Check‘, welche nach der Instanziierung der Klasse durch das Hauptprogramm aufgerufen wird. Sie bekommt als Parameter eine Auftragsnummer übergeben und besitzt keinen Rückgabewert. Sie ist der Einstiegspunkt des NuGet Pakets. Danach wurden 2 Services angelegt, welche die Geschäftslogik enthalten. Der ‚PTKAPIService‘ enthält eine einzige Methode ‚GetFlyDBItem‘ mit der Auftragsnummer als

Parameter. Diese Methode führt einen GET-Request an die PTKAPI durch, welcher nach erfolgreichem Abschluss, alle Auftragsdaten zu diesem Auftrag in der Response enthält. Der relevante Teil dieser Daten wird dann mithilfe eines Tuples von der Methode zurückgegeben. Der zweite Service ‚APIService‘ besitzt drei Methoden und kommuniziert mit der ‚PackageServiceAPI‘. Eine Methode ‚CheckForSurvey‘ nimmt die ProductVariantId, die Quantity und die UserId als Parameter entgegen, welche dann als QueryParameter im GET-Request an den Endpunkt ‚api/product/survey‘ gesendet werden. Falls eine Umfrage durchgeführt werden soll, wird die ProductId des Datensatzes aus der Datenbank zurückgegeben. Ansonste ist der Rückgabewert ‚null‘. Eine zweite Methode holt sich von der API alle Verpackungen, welche in der Datenbank hinterlegt sind und gibt diese zurück. Die dritte und letzte Methode, bekommt eine Liste der vom Benutzer angegebenen Verpackungen als Übergabeparameter und sendet eine POST-Request an die API. Wenn jetzt die oben genannte ‚Check‘ Methode aufgerufen wird, wird mithilfe der Methoden dieser beiden Services ermittelt, ob für die angegebene Auftragsnummer bereits alle Umfragen durchgeführt wurden und es eine fest hinterlegte Verpackung gibt. Ist das nicht der Fall, wird ein Umfragefenster geöffnet, welches den Benutzer dazu auffordert, seine verwendete Verpackung anzugeben.

5.4.3 Erstellen der Benutzeroberfläche

Nachdem die Logik implementiert war, wurde die Benutzeroberfläche umgesetzt. Diese verwendet, wie bereits oben erwähnt, die Windows Presentation Foundation (WPF). Ein WPF-Window besteht dabei immer aus zwei zusammenhängenden Teilen, eine ‚.xaml‘, in welcher das Layout der Seite erstellt wird, und eine ‚.xaml.cs‘, in welcher die Logik zur Seite angelegt wird. Dieser zweite Teil wird auch Codebehind genannt. Darin werden alle Events, wie z.B. ein Klick auf einen Knopf ausformuliert. Das Design der Umfrageseite ist in XAML geschrieben. Hierbei handelt es sich um eine auf XML basierende Sprache, welche zum Erstellen des Layouts der Benutzeroberfläche verwendet wird. Die Seite besteht aus einem Dockpanel, welches zwei Groupboxen beinhaltet. Die obere zeigt die ausgewählten Verpackungen, wohingegen die untere die Komponenten zum Hinzufügen einer neuen Verpackung und den Knopf zum Absenden der Umfrage umfasst. Die Schaltflächen werden mithilfe einer Datenbindung an das ViewModel gebunden. Bei einem ViewModel handelt es sich um die Sammlung aller Models für die Benutzeroberfläche, welches bei der Instanziierung des Fensters als DataContext hinzugefügt wird (siehe Abbildung 6). Das heißt, hier werden alle relevanten Daten gesammelt.


```
internal class SurveyViewModel
{
    1 Verweis
    public List<Package> Packages { get; set; }
    0 Verweise
    public ObservableCollection<PackageItem> PackageItems { get; set; }
}
```

Abbildung 6: ViewModel

6 Testen und Abnahme

6.1 Test durch den Entwickler

Alle Anwendungen wurden vom Autor durch Schreibtischtests und Debugging getestet. Allerdings weicht bei Javascript das Debugging ein wenig ab. Hier wurden die aktuellen Werte der Variablen in die Konsole ausgegeben und verglichen, da Haltepunkte nicht funktionieren.

6.1.1 Code-Review

Um während der Erstellung des Projektes Meinungen und Anregungen von Dritten einzuholen, wurde der Code in unterschiedlichen Phasen der Entwicklung per Code-Review anderen Programmierern vorgestellt. Das dadurch gesammelte Feedback konnte dann zur Verbesserung der zu diesem Zeitpunkt aktuellen und zukünftigen Programmabläufe verwendet werden.

6.1.2 Schreibtischtests

Mit Hilfe von Schreibtischtest war es dem Autor möglich, sich individuelle Teile des Codes herauszugreifen und zu überlegen, ob die verwendete Logik das gewünschte Ergebnis liefert. Diese Tests wurden hauptsächlich während der Entwicklungsphase bei komplexeren Abfragen oder Algorithmen eingesetzt.

6.1.3 Debugging

Das Debugging wurde am häufigsten zum Testen der Anwendung verwendet. Dabei wurden im Code an bestimmten Stellen Haltepunkte eingefügt, um zu schauen, ob die verwendeten Variablen an dieser Stelle die vorhergesehenen Werte besitzen. Außerdem kann die Anwendung jederzeit beendet oder Zeile für Zeile ausgeführt werden, um mögliche Fehler in Abläufen zu erkennen. Der große Vorteil dabei ist, dass nicht jedes Mal die komplette Anwendung durchlaufen werden muss, was viel Zeit spart. Mit diesem Verfahren wurden zum Beispiel die an die API gesendeten Daten überprüft, ohne dass diese in die Datenbank geschrieben wurden.

6.2 Abnahme durch Teamleiter

Die Abnahme durch den Teamleiter erfolgte durch die Vorstellung aller Komponenten des Projektes. Anschließend wurde über mögliche Verbesserungen und Anpassungen geredet, welche möglicherweise in Zukunft noch getätigt werden müssen.

7 Dokumentation

7.1 REST-Schnittstelle

Die Schnittstellendokumentation wurde durch das Swagger Framework erstellt. Durch das Paket werden dem Benutzer die Routen und die benötigten Parameter grafisch dargestellt. Außerdem können über diese Oberfläche auch direkt alle Routen getestet werden. Eine Entwicklerdokumentation erfolgt wo nötig per Kommentaren im Code.

7.2 Dashboard

Für das Dashboard wurde eine Anwenderdokumentation erstellt, da die Webseite in Zukunft von ausgewählten Personen zum Administrieren der verwendeten Verpackungen eingesetzt werden soll. Hier wurde ein Screenshot der Seite erstellt und die wichtigsten Schaltflächen erklärt, zu sehen in Anhang 7. Eine Entwicklerdokumentation erfolgte wie bei der REST API mit Kommentaren im Code.

7.3 NuGet-Paket


Da auch das NuGet-Paket eine Benutzeroberfläche besitzt, wurde auch hierfür ein Screenshot erstellt und die wichtigsten Schaltflächen beschriftet. Die Anwenderdokumentation ist hierbei für die Mitarbeiter der Produktion, welche dann die Umfragen ausführen müssen. Diese ist in Anhang 8 zu sehen. Auch für das Paket wurde eine Entwicklerdokumentation anhand von Kommentaren im Code ausgeführt.

8 Fazit

Der Zeitplan von 70 Stunden konnte eingehalten werden. Während der Entwicklung traten keine größeren oder unerwarteten Probleme auf, sodass das Projekt reibungslos durchgeführt werden konnte. Die verwendeten Pakete waren dem Autor bereits gut bekannt und werden auch in weiteren Projekten dieser Art zum Einsatz kommen. Vor allem das Paket Mantine hat überzeugt, da mithilfe der Komponenten schnell und einfach schöne Webseiten in React erstellt werden können.

Glossar

Sammeldruckverfahren

Ein Produktionsverfahren beim dem versucht wird Aufträge verschiedener Kunden mit dem gleichen Papier zusammen auf einen Bogen zu platzieren. Dadurch führt es zu weniger Weißfläche, wodurch die Kosten und damit der Preis gesenkt werden kann. 

NuGet

NuGet ist ein Paket Manager für .NET, mit dessen Hilfe schnell und einfach wiederverwendbarer Code per NuGet Paket anderen Entwicklern zur Verfügung gestellt werden kann. Diese müssen das Paket lediglich über den Manager installieren.

Crowdsourcing

Die Auslagerung einer Aufgabe an eine Gruppe von Menschen.

Importer

hier: Anwendung, welche Daten für ein System importiert.

NoSQL Datenbanken

nicht relationale Datenbanken

Windows Presentation Foundation

Ein Benutzeroberflächen-Framework für .NET mit dessen Hilfe Desktopclientanwendungen erstellt werden können. Als Markup Language wird XAML verwendet.

REST-Schnittstelle / REST API

Eine Programmierschnittstelle die nach der Representational State Transfer Architektur erstellt wurde.

API

Eine Anwendungsschnittstelle, welche von anderen Anwendungen angesprochen werden kann.

Docker Swarm

Ein Cluster aus Docker Servern.

Progressive Web App

Eine Website, welche sich wie eine App installieren lässt.

Active Directory

Verzeichnisdienst von Windows

JWT Token

Ein genormter Access-Token, um den sicheren Austausch von Benutzerinformationen zu gewährleisten

Bearer Token

Ein Token zur Authentifizierung in ‚Authorization‘ Headers, welcher den eigentlichen JWT Token enthält

Dependency Injection

Ein Entwurfsmuster, bei dem ein Objekt an einem zentralen Punkt initialisiert wird und dann den Klassen im Konstruktor zur Verfügung steht.

**CRUD-Routen**

Routen für Create, Read, Update und Delete Operationen in der Datenbank

**NPM**

Paket Manager für node, welcher per Konsole Pakete verwalten kann.

SessionStorage

Ein Speicherplatz im Browser, um während einer Session Daten zu speichern. Wird nach dem Schließen der Seite gelöscht.

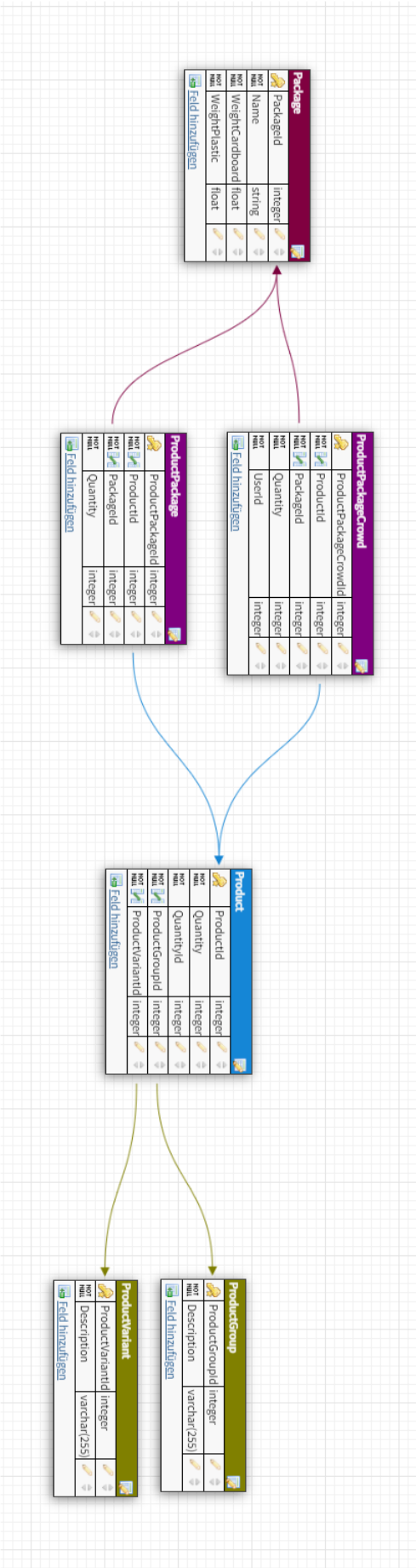
Abkürzungsverzeichnis

FA	Flyeralarm GmbH
FAIP	FLYERALARM Industrial Print GmbH
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
ORM	Object Relational Mapper
PWA	Progressiv Web App
B2B	Business to Business
FTP	File Transfer Protocol
REST	Representational State Transfer
ASP	Active Server Pages
EF Core	Entity Framework Core
JWT	JSON Web Token
SQL	Structured Query Language
CRUD	Create, Read, Update und Delete
Npm	Node Package Manager
API	Application Programming Interface

Anhang 1: Projektphasen

1. Analyse	3 h
1.1 Ist-Analyse	1 h
1.2 Wirtschaftlichkeitsanalyse	1 h
1.3 Soll-Konzept	1 h
2. Entwurf	4 h
2.1 Entwurf der Datenbankstruktur	2 h
2.2 Mockup Benutzeroberfläche	2 h
3. Implementierung	38 h
3.1 Erstellen der Datenbank	3 h
3.2 Erstellen der Rest API	12 h
3.2.1 Datenbankanbindung	2 h
3.2.2 Routenimplementierung	9 h
3.2.3 Schnittstellendokumentation mit Swagger	1 h
3.3 Erstellen des Dashboards	14 h
3.3.1 Anbindung an Rest API	3 h
3.3.2 Erstellen der Benutzeroberfläche	8 h
3.3.3 Erstellen der Authentifizierung	3 h
3.4 Erstellen des NuGet Pakets	9 h
3.4.1 Implementierung der Logik	5 h
3.4.2 Erstellen der Benutzeroberfläche	4 h
4. Test	5 h
4.1 Code-Review	2 h
4.2 Schreibtischtests	3 h
5. Einführung	4 h
5.1 Inbetriebnahme der API und des Dashboards	3 h
5.2 Abnahme durch Teamleiter	1 h
6. Dokumentation	16 h
6.1 Erstellen der Entwicklerdokumentation	3 h
6.2 Erstellen der Anwenderdokumentation	3 h
6.3 Erstellen der Projektdokumentation	10 h
Summe	70 h

Anhang 2: Datenbankmodell



Anhang 3: Mockup Dashboard

PackageService

- Theme umschalten
- Abmelden

ProductVariantId	Quantity	QuantityId	ProductGroupId	Status	
6543	100	1234	5890	1 / 10	bearbeiten
6543	1000	1234	5890	3 / 10	bearbeiten
4587	500	8969	5890	10 / 10	bearbeiten

<- 1 2 3 ... 10 ->

Anhang 4: Ressourcenplanung

Art	Bezeichnung
Personal	Entwickler / Auszubildender
	Teammitglieder
	Teamleiter / Projektbetreuer
	Product Owner
Hardware	HP Z2 Mini G4 Workstation
	Server
Software	Windows 10
	Visual Studio 2022
	Visual Studio Code
	DbGate / DBeaver
	Docker Desktop
	Microsoft Word

Anhang 5: Ausschnitt Swagger-Dokumentation

Product

GET `/Product` Get all products

POST `/Product` Post new product

GET `/Product/survey` Check if survey is needed

Parameters

Name	Description
productVariantId integer(\$int32) (query)	<input type="text" value="productVariantId"/>
quantity integer(\$int32) (query)	<input type="text" value="quantity"/>
userId integer(\$int32) (query)	<input type="text" value="userId"/>

Responses

Code	Description	Links
200	Success	No links

GET `/Product/{id}` Get product by id

PUT `/Product/{id}` Update product by id

DELETE `/Product/{id}` Delete product by id

PATCH `/Product/{id}/packages` Override packages of product

Anhang 6: Finales Design des Dashboards im Darkmode

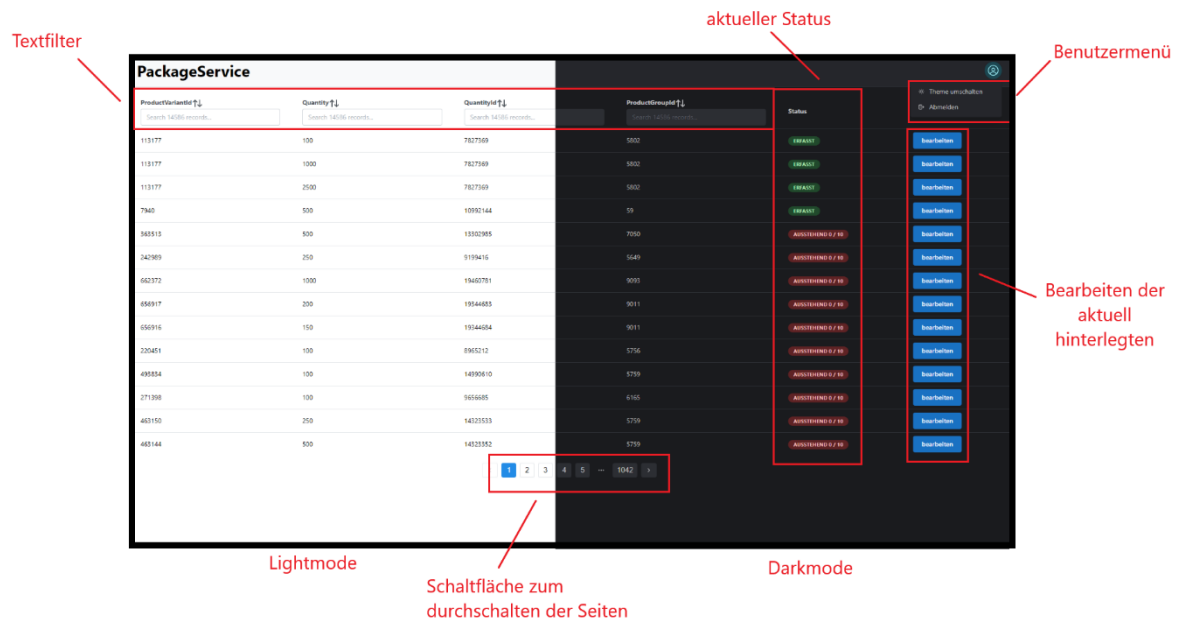
PackageService

⊗ Theme umschalten
👤 Abmelden

ProductVariantId	Quantity	QuantityId	ProductGroupid	Status	
113177	100	7827369	5802	ERFASST	bearbeiten
113177	1000	7827369	5802	ERFASST	bearbeiten
113177	2500	7827369	5802	ERFASST	bearbeiten
7940	500	10992144	59	ERFASST	bearbeiten
363513	500	13302985	7050	AUSSTEHEND 0 / 10	bearbeiten
242989	250	9199416	5649	AUSSTEHEND 0 / 10	bearbeiten
662372	1000	19460781	9093	AUSSTEHEND 0 / 10	bearbeiten
656917	200	19344683	9011	AUSSTEHEND 0 / 10	bearbeiten
656916	150	19344684	9011	AUSSTEHEND 0 / 10	bearbeiten
220451	100	8965212	5756	AUSSTEHEND 0 / 10	bearbeiten
493834	100	14990610	5759	AUSSTEHEND 0 / 10	bearbeiten
271398	100	9656685	6165	AUSSTEHEND 0 / 10	bearbeiten
463150	250	14323533	5759	AUSSTEHEND 0 / 10	bearbeiten
463144	500	14323552	5759	AUSSTEHEND 0 / 10	bearbeiten

1 2 3 4 5 ... 1042 >

Anhang 7: Anwenderdokumentation Dashboard



Anhang 8: Anwenderdokumentation NuGet Umfrage

